Vol.2018-SE-198 No.23
2018/3/10

再利用のためのWebページからのHTML/CSSテンプレート生成

額田 蓮[1,a)]　大久保 弘崇[2,b)]　粕谷 英人[2,c)]　山本 晋一郎[2,d)]

概要：本研究は Web ページから任意の領域を切り出す手法を提案する．切り出しは HTML/CSS 形式のままで行うため，開発者が既存の Web ページの内容を再利用してテンプレートを生成する作業を支援することができる．切り出す範囲の指定は HTML の要素単位で行い，切り出す範囲に適用される CSS 規則のみを抽出する．また，切り出しにより HTML の文書構造が変化するため，切り出しの前後でスタイルの同一性を保つために HTML に依存した CSS コードを編集する．

1. はじめに

1.1 背景

既存の Web ページの HTML/CSS の一部分を他の Web ページで再利用することで Web 開発の労力を低減することができる．繰り返し使う場合にはこれをテンプレートとして保存することで，同じようなコードを再び書く必要がなくなる．

Web ページから指定する任意の領域だけを切り出すには，指定した領域に対応した HTML/CSS のソースコードを抽出する．しかし，CSS によるスタイルの記述には HTML の構造に依存する記法がある．そのため，HTML の部分的な切り出しによる文書構造の変化は，CSS のスタイル適用にも影響を及ぼすため Web ページの体裁が崩れてしまう原因となる．指定した領域に対応する HTML とその部分に適用される CSS を検出し，抽出するだけではスタイルを維持した Web ページの切り出しは行えない．このような HTML と CSS の対応関係を確認しながら切り出しを手動で行うためには，HTML/CSS に関する知識が必要となる．これは HTML/CSS のソースコードの量が多くなるにつれて労力は大きくなる．

[1] 愛知県立大学大学院情報科学研究科
Graduate School of Information Science and Technology, Aichi Prefectural University
[2] 愛知県立大学情報科学部
School of Information Science and Technology, Aichi Prefectural University
[a)] nukada@yamamoto.ist.aichi-pu.ac.jp
[b)] ohkubo@ist.aichi-pu.ac.jp
[c)] kasuya@ist.aichi-pu.ac.jp
[d)] yamamoto@ist.aichi-pu.ac.jp

1.2 目的

本研究では，Web ページからスタイルを維持して任意の領域を切り出す手法を提案する．切り出す領域は HTML の要素単位で指定する．切り出しは HTML/CSS コードの抽出とスタイル維持のためのコード変換により行う．既存の Web ページから表示を変えることなく任意の領域を切り出すことで HTML テンプレートの作成を支援する．

1.3 貢献

本研究は HTML の文書構造の変化に伴うスタイルの変化を考慮した切り出し方法を提案した．これにより Web ページから任意の領域をスタイルを変えることなく HTML/CSS コードの抽出が可能となる．

2. 切り出しによるWebページのスタイル崩れ

2.1 Web ページの構成

Web ページは HTML と CSS に加え，画像などのリソースで構成される．HTML はタグを用いて記述するツリー構造のマークアップ言語である．タグは開始タグと終了タグを用いて記述される．開始タグと終了タグで括られた全体を要素と呼ぶ．

CSS は HTML などで記述されたドキュメントのフォントや色などの体裁を記述するためのスタイルシート言語である．CSS は HTML の要素をどのように表示するかを定義する CSS 規則の集合である．CSS 規則は HTML の要素を指定するセレクタとその要素にどのようなスタイルを適用するかを記述する宣言ブロックで構成される．宣言ブロックはプロパティとその値の組の集合で構成される．

2.2 HTML に依存した CSS 記述

CSS には HTML に依存した記述が存在する．セレクタには HTML の要素を直接指定する方法や他の HTML の要素との相対的な構造関係により指定する方法などがある．また，プロパティにはサイズなどを絶対値で指定する方法と親要素との相対的値で指定する方法がある．これらの HTML に依存した CSS 記述は，HTML の部分を切り出すことでスタイルの適用に影響が出る．2.2.1 節と 2.2.2 節で HTML に依存したセレクタとプロパティについてそれぞれ述べる．

2.2.1 セレクタ

CSS3 には 42 種類のセレクタが定義されている [1]．それらは以下の 5 つのカテゴリに分類される．

- 簡易セレクタ
- 属性セレクタ
- 結合子
- 擬似クラス
- 擬似要素

擬似クラスと結合子には他要素との関係を利用するものがある．例えば結合子の一つの E+F セレクタは，要素 E の直後に後続する要素 F を指定する．E+F セレクタは他要素との関係に依存しており，HTML の構造が変化することで指定する要素が変化してしまう可能性がある．元の HTML コードではセレクタの指定対象であった要素が，切り出した HTML コードでは指定外になる場合と逆に指定外であった要素が指定される場合がある．それぞれの場合について，E+F セレクタを例にリスト 1，リスト 2 を用いて説明する．リスト 2 はファイル名を main.css とし，リスト 1 の link タグを用いて参照している．リスト 2 の h1 + ⋆は h1要素の直後に後続する任意の要素を指定する．そのため，リスト 1 の h1 要素の直後に現れる img 要素に width プロパティの値を 300px にするスタイルが適用される．Web ブラウザでこの HTML ファイルを表示したものを図 1 に示す．HTML の構造の変化により Web ページの体裁がどのように変化するかを図 1 と比較して説明していく．

セレクタの指定対象外になる場合

切り出しによって要素がセレクタの指定対象外になる場合について考える．リスト 1 の 7 行目にある h1 要素を削除し，8 行目の img 要素と 9 行目から 11 行目の p 要素が切り出されるとする．h1 + ⋆セレクタにより h1 要素の直後に後続する img 要素にスタイルが適用されていたが，h1 要素がなくなったことで img 要素はセレクタの指定対象から外れる．通常，img 要素の width プロパティの値は指定がない場合に画像ファイルの横幅が設定される．よって，width プロパティで指定していた大きさと実際の画像ファイルの横幅の大きさが異なる場合に img 要素のサイズが変化してしまう．Web ブラウザで表示したものを図 2 に示す．図 1 と比較して，width プロパティの値が 300px に指定されなくなったため画像のサイズが変化している．

セレクタの意図しない指定がされた場合

切り出しによってセレクタの指定対象外であった要素が指定される場合について考える．リスト 1 の 8 行目にある img 要素を削除し，7 行目の h1 要素と 9 行目から 11 行目

リスト 1　HTML のサンプルコード 1

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <link rel="stylesheet" href="main.css"/>
5    </head>
6    <body>
7      <h1>Lorem ipsum</h1>
8      <img src="image.png">
9      <p>
10       vulputate eget, arcu. In enim justo,
           rhoncus ut, imperdiet a, venenatis
           vitae justo. Nullam dictum felis eu
            pede mollis.
11     </p>
12   </body>
13 </html>
```

リスト 2　CSS のサンプルコード 1

```
1 h1 + * {
2     width : 300px;
3 }
```

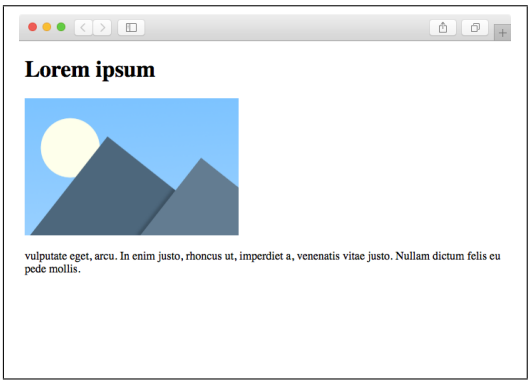

図 1　リスト 1 の表示結果

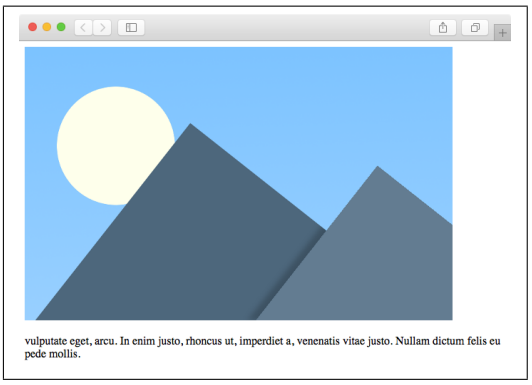

図 2　セレクタの指定対象外になった場合の表示結果

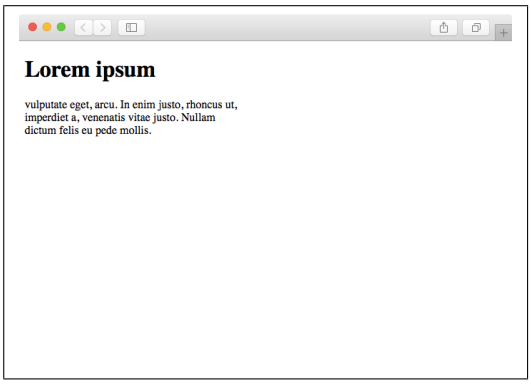

図 3　意図しない指定がされた場合の表示結果

リスト 3　HTML のサンプルコード 2

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <link rel="stylesheet" href="main.css"/>
5    </head>
6    <body>
7      <div>
8        <h1>Donec quam</h1>
9        <p>
10         Nulla consequat massa quis enim.
               Donec pede justo, fringilla vel,
               aliquet nec vulputate eget, arcu.
                In enim justo.
11       </p>
12     </div>
13   </body>
14 </html>
```

の p 要素が切り出されるとする．img 要素がなくなったことで h1 要素の直後に p 要素が後続している．そのため，h1 + *セレクタにより p 要素にスタイルが適用されることになる．これは HTML の構造の変化前とは異なる要素への意図しないスタイルの適用である．Web ブラウザでこれを表示したものを図 3 に示す．図 1 と比較して，width プロパティの値が 300px に指定されたため p 要素の文章の横幅が変化している．

2.2.2　プロパティ値

CSS プロパティの値には親要素や兄弟要素のプロパティ値に依存して値が定まる相対的な指定ができる．width プロパティなどでは 300px のような絶対値の他に 50%のような値を指定できる．この場合，親要素を 100%としたときの相対的な値となる．そのため，親要素が変更された場合に相対的な値の算出値が変わる．リスト 3，リスト 4 を例に説明する．Web ブラウザでこの HTML ファイルを表示したものを図 4 に示す．

このとき p 要素の width の算出値は $500 \times 0.5 \times 0.8 = 200$px となる．プロパティ値に親要素に対する相対的な指定がされているため，切り出しによって親要素が無くなったときに算出値が変化してしまう．p 要素を切り出し，親

リスト 4　CSS のサンプルコード 2

```
1  body {
2      width : 500px;
3  }
4  div {
5      width : 50%;
6  }
7  p {
8      width : 80%;
9  }
```

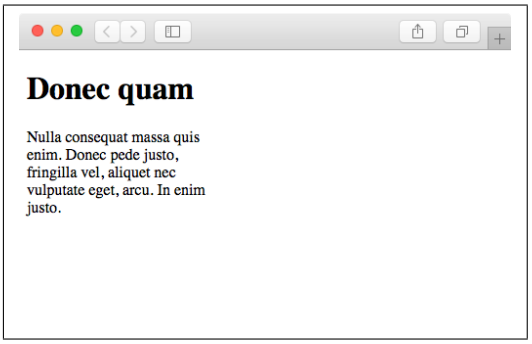

図 4　リスト 3 の表示結果

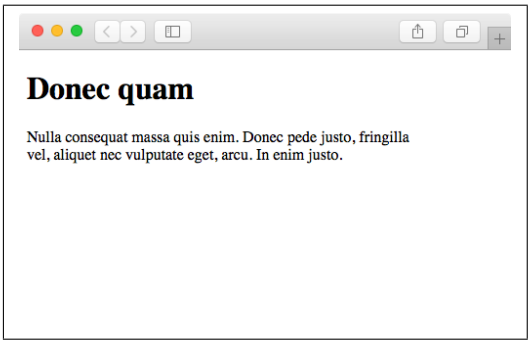

図 5　相対値が変化する例の表示結果

要素である div 要素がなくなった場合について考える．div 要素のタグであるリスト 3 の 7 行目と 12 行目を削除する．これにより div 要素の子要素だった h1 要素と p 要素は body 要素の子要素となる．このとき p 要素の width の算出値は $500 \times 0.8 = 400$px となる．Web ブラウザでこれを表示したものを図 5 に示す．図 4 と比較して，width プロパティの値が 200px から 400px に変わったため，p 要素の文章の横幅が変化している．このようにプロパティの値に相対値を用いた場合は HTML 構造の変化によって値が変わってしまう．

3.　提案手法

本研究では Web ページからユーザーが指定した領域に対応した HTML/CSS のソースコードを切り出して，HTML に依存した CSS のための変換をすることで，HTML テンプレートを作成する手法を提案する．ユーザーが指定した領域と対応する HTML の切り出しとその部分に適用される CSS の切り出しについて 3.1 節で述べる．切り出した CSS

の変換方法について 3.2 節で述べる．本手法は HTML4.01，HTML5 と CSS2.1，CSS3 を対象とする．

3.1 HTML/CSS の切り出し

指定領域の HTML コードとそれに適用されている CSS 規則を切り出す方法について述べる．

HTML

まず，指定した Web ページの領域に対応する HTML を切り出す方法について述べる．HTML の切り出しは HTML の要素単位で行う．ユーザーが指定した領域と対応する HTML の要素を切り出しの対象とする．HTML のタグを用いたツリー構造は DOM ツリーとして表現されており，HTML の要素は DOM ツリーのノードと一対一で対応する．DOM は W3C 勧告により仕様が策定されている HTML のための API である [2]．Firefox や Google Chrome などの多くのブラウザでは DOM API が実装されており，JavaScript を用いて DOM ツリーに対する操作が可能である．HTML 要素の移動や削除，ソースコードの取得などの操作は DOM API を用いることで実現する．

ユーザーは切り出す領域に該当する要素を選択する．ただし，本手法では Web ページの表示を変えることなく切り出しを行うことが目的であるため，横幅を維持できない要素は切り出しの対象としない．HTML の要素は表示の役割によってブロックレベル要素とインラインレベル要素に大きく分けられる．これは `display` プロパティの値によって決定する．ブロックレベル要素は新しい行から始まるのに対して，インラインレベル要素は行を分断せず横幅は必要な幅のみを占有する．そのため，インラインレベル要素は `width` プロパティなどによるサイズの指定ができない．ただし，インラインレベル要素のうち `img` 要素などのインライン置換要素はサイズの指定ができる．それらは固有の高さや幅を持っており，画像であれば特定のピクセル数に設定される．よって，ユーザーが指定できる要素はブロックレベル要素とインライン置換要素である．なお，要素を選択する方法は様々あるが，4 章で述べる実装では DOM API のマウスイベントを用いてマウスカーソルにより選択している．

HTML コードの切り出しを次の手順で DOM API を用いて DOM ツリーへの操作をすることで行う．ユーザーが指定した切り出しを行う DOM ツリーのノードの集合を SN とし，HTML の `body` 要素の子ノードの集合を BC とする．SN の各ノードは互いに親または祖先の関係にないこととする．HTML のタグがソースコードに記述されている順番に SN からノードを取り出し，`body` 要素の末尾の子ノードとなるよう DOM ツリーに挿入していく．DOM ノードの移動をするため，HTML コードでは切り出すノードのタグ部分は削除され，`body` タグの入れ子として末尾に切り出すノードのタグ部分が挿入されることになる．次に，$BC - SN$ のノードをすべて削除する．SN のノードは BC のノードを親または祖先に持つか自分自身であるため挿入操作を行ってから $BC - SN$ のノードを削除した．

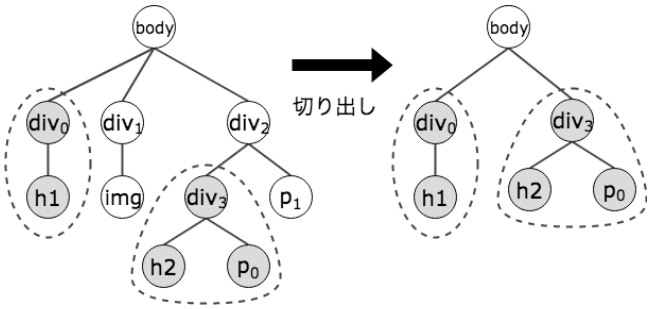

図 6 HTML の切り出しによる DOM ツリーの変化

図 6 に切り出しによる DOM ツリーの変化の例を示す．このとき，$SN = \{\mathtt{div_0}, \mathtt{div_3}\}$，$BC = \{\mathtt{div_0}, \mathtt{div_1}, \mathtt{div_2}\}$ である．SN から $\mathtt{div_0}, \mathtt{div_3}$ の順に取り出し，`body` 要素の末尾の子ノードとなるように挿入する．よって，$BC = \{\mathtt{div_1}, \mathtt{div_2}, \mathtt{div_0}, \mathtt{div_3}\}$ となる．最後に，$BC - SN = \{\mathtt{div_1}, \mathtt{div_2}\}$ を削除する．これにより，ユーザーが選択した SN のみ `body` 要素の子要素として残る．

CSS

切り出した HTML に適用される CSS を切り出す方法について述べる．CSS ファイルに含まれる各 CSS 規則は DOM API を用いて取得できる．取得した CSS 規則から，どの CSS を切り出すかは次のように決める．SN のノードとそれを親または祖先に持つ全てのノードの集合を SN_{all} とする．ある CSS 規則のセレクタが指定するノードの集合を CN とする．$SN_{all} \cap CN$ が空集合でないとき，その CSS 規則は切り出した HTML に適用されているため切り出す対象とする．全ての CSS 規則に対してこれを行う．

図 6 を用いて切り出す CSS 規則の検出の例を説明する．図 6 の DOM ツリーが表す HTML コードに適用する CSS を以下に示す．1 行目から 3 行目の CSS 規則を `CSSRule1`，4 行目から 6 行目の CSS 規則を `CSSRule2` とする．

```
1 p {
2   color : red;
3 }
4 img {
5   width : 500px;
6 }
```

$SN_{all} = \{\mathtt{div_0}, \mathtt{h1}, \mathtt{div_3}, \mathtt{h2}, \mathtt{p_0}\}$ であり，`CSSRule1` が指定するノードの集合は $CN = \{\mathtt{p_0}, \mathtt{p_1}\}$ である．よって，$SN_{all} \cap CN = \{\mathtt{p_0}\}$ である．次に，`CSSRule2` が指定するノードの集合は $CN = \{\mathtt{img}\}$ であるから，$SN_{all} \cap CN$ は空集合である．よって，`CSSRule1` を切り出すが，`CSSRule2` は切り出さない．

擬似要素セレクタ

上で述べた，切り出す CSS 規則の適用対象となる HTML

の要素はDOM APIのquerySelectorAll関数により取得できる．しかし，この関数は仕様により擬似要素に対応していないという問題がある．擬似要素セレクタとは選択された要素の特定の部分に対してスタイルを適用するものである．擬似要素セレクタの一つであるE::first-letterはE要素が持つテキストの最初の文字にスタイルを適用する．例としてE::first-letterを用いたHTMLとCSSを以下に示す．このとき，pタグのテキストHello, World!!の先頭文字であるHだけが赤色となる．

```
<p> Hello, World!! </p>
```

```
p::first-letter {
  color : red;
}
```

このように擬似要素セレクタは架空の要素を指定するため，HTMLのソースコードに擬似要素は存在しない．この架空の要素p::first-letterは次のように考えることができる．

```
<p> <p::first-letter>H</p::first-
    letter>ello, World!! </p>
```

要素p::first-letterはp要素に内包されており，この擬似的なスタイルの適用は実質的にp要素に対するスタイルの適用であり，その先頭文字にのみ効果が現れる．よって，CSS規則が適用される擬似要素を取得するためにはE::first-letterから擬似要素の記述を除いたEセレクタを用いてquerySelectorAll関数により取得する．他の擬似要素セレクタに対しても同様の方法を用いる．

擬似クラスセレクタ

擬似クラスセレクタはHTMLの要素が特定の状態であることを指定するものがある．例えば擬似クラスセレクタの一つであるE::hoverはマウスカーソルがE要素の上にある場合にのみ指定する．そのため，擬似クラスセレクタによって指定されるHTMLの要素は特定の状態のときにしかquerySelectorAll関数で取得できない．このように特定の状態時にのみスタイルが適用されるが，E::hoverによって指定される要素自体はE要素である．よって，擬似クラスセレクタが指定する要素を取得するためにはE::hoverから擬似クラスの記述を除いたEセレクタを用いてquerySelectorAll関数により取得する．このような外的要因に状態が影響する擬似クラスセレクタには同様の方法を用いる．

3.2 スタイル維持のためのCSSコード変換

HTMLに依存したCSSのセレクタとプロパティ値に対するスタイル維持のためのコード変換について述べる．

3.2.1 セレクタ

切り出しの前後でセレクタが指すHTMLの要素を比較し，スタイルの適用範囲が変化したCSS規則の検出とそのスタイル適用の修正を行う．

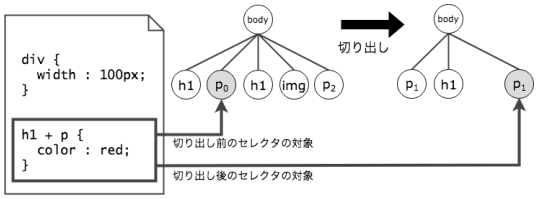

図 7　切り出しの前後でのセレクタの対象の変化

スタイル適用範囲の変化の検出

スタイル適用範囲の変化にはセレクタの指定対象外になる場合とセレクタの意図しない指定がされた場合がある．切り出しの前後でセレクタが指定するDOMノードを比較してそれぞれの場合の変化を検出する．切り出し前のセレクタが指定するノードの集合をBN，切り出しの後にセレクタが指定するノードの集合をANとする．$BN - AN$をセレクタの指定対象外になるノードとして検出し，$AN - BN$をセレクタの意図しない指定がされたノードとして検出する．

図7にHTMLの切り出しの前後におけるセレクタの対象の変化の例を示す．これを用いてスタイル適用範囲の変化の検出について具体例を述べる．便宜的に図の2つのp要素を区別するためにp_0，p_1と記述する．切り出し前にh1 + pセレクタによってスタイルが適用される要素はp_0である．よって，h1 + pセレクタが指すノードの集合は$BN = \{p_0\}$となる．切り出し後はp_0の前のh1要素がなくなることで，p_0はh1 + pセレクタの対象ではなくなる．また，p_1は一つ前のimg要素がなくなることで二つ前のh1要素と隣接する．これにより元々対象ではなかったp_1が切り出しによりh1 + pセレクタの対象となる．よって，$AN = \{p_1\}$となる．$BN - AN = \{p_0\}$であるからp_0をセレクタの指定対象外になるノードとして検出し，$AN - BN = \{p_1\}$であるからp_1をセレクタの意図しない指定がされたノードとして検出する．

スタイル適用の修正

検出したDOMノードに対してスタイル適用の修正を行う．最初に，セレクタの指定対象外になる場合の修正ついて述べる．検出したセレクタの指定対象外になる要素をN_{out}とする．そのセレクタのCSS規則を$CSSRule_{out}$とする．N_{out}に対して，$CSSRule_{out}$を再適用させる．再適用するためにはN_{out}を指定するようなセレクタを$CSSRule_{out}$のセレクタに追加する．なお，複数の異なるセレクタをコンマで区切って記述することで同じCSS規則を適用できる．

このセレクタは他の要素に影響を与えないようにN_{out}を一意に指定する必要がある．要素を一意に指定するセレクタにはidセレクタがある．しかし，本手法ではidセレクタの使用を避け，E+Fと:nth-child(n)を組み合わせたセレクタを用いて要素のパスから一意に指定する．

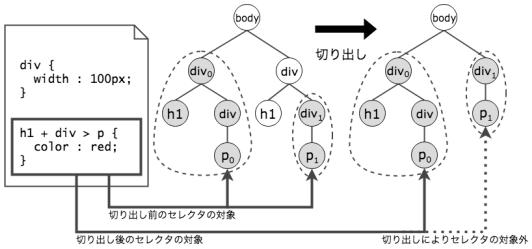

図 8 パスセレクタを用いたスタイル再適用の例

本手法ではこれをパスセレクタと呼ぶこととする．

図 8 を用いて，パスセレクタの具体例とパスセレクタを用いてセレクタの指定対象外となった要素に CSS 規則を再適用する例について説明する．まず，パスセレクタの具体例について述べる．図 8 の p_0 要素を指定するパスセレクタは `body>div:nth-child(1)>div:nth-child(2)>p:nth-child(1)` となる．`E>F` セレクタはある要素 `E` の直接の子要素であるような要素 `F` を指定する．整数 `n` を指定した`:nth-child(n)` セレクタはある親要素の `n` 番目の子要素である要素 `E` を指定する．

本手法でパスセレクタを使用する際には，パスセレクタの開始を `body` 要素をルートとする絶対パスセレクタではなく，切り出しのためにユーザーが指定した *SN* の要素をルートとする相対パスセレクタを用いる．*SN* の要素が持つ `id` を用いて，その要素を指定する `id` セレクタをルートとした相対パスセレクタを生成する．要素が `id` を持たない場合は，重複しない固有 `id` を与える．*SN* の要素だけに `id` を与え，その要素の内部要素に対する修正には相対パスセレクタを用いることで `id` の多用を避ける．

次に，セレクタの指定対象外となった要素に CSS 規則を再適用する方法について述べる．図 8 の `h1+div>p` セレクタによる CSS 規則を考える．切り出しによって `h1+div>p` セレクタの指定から外れた要素 p_1 をパスセレクタを用いてスタイルの再適用をする．p_1 を N_{out} とする．`h1+div>p` セレクタによる CSS 規則を $CSSRule_{out}$ とする．$SN = \{div_0, div_1\}$ それぞれの要素に `id` として `"ID000"`, `"ID001"`を与える．このとき，N_{out} を指定するパスセレクタは`#ID001>p:nth-child(1)` となる．このパスセレクタを $CSSRule_{out}$ のセレクタに追加することで，N_{out} に再び $CSSRule_{out}$ を適用する．よって，$CSSRule_{out}$ は次のような CSS 規則となる．2 行目が本手法によって追加された再適用のためのセレクタである．

```
1 h1 + div > p,
2 #ID001 > p:nth-child(2)
3 {
4   color : red;
5 }
```

次に，セレクタの意図しない指定がされた場合の修正について述べる．意図しないスタイルの適用がされた要素を N_{in} とする．N_{in} だけを例外的にセレクタの対象から除くには，`E:not(F)` セレクタを用いる方法がある．しかし，後述の理由により本手法では `E:not(F)` セレクタを使用できない．そのため，意図しない適用が行われたスタイルを打ち消すための CSS 規則を追加するものとする．打ち消し CSS 規則のセレクタは N_{in} のパスセレクタとする．打ち消しのためのプロパティ値には，切り取り前の N_{in} の算出スタイルを利用する．算出スタイルとは要素に適用される全てのプロパティが最終的に算出されたものである．算出スタイルは `getComputedStyle`関数によって取得することができる．

本手法では次のような理由により `id` の多用を避ける．`id` セレクタで要素を指定するには，`id` を要素に対して一意に割り当てる必要がある．そのため本手法の再適用のためだけに与える `id` はコードの再利用性を下げる．また，`ID000`, `ID001`, ... のような自動生成される重複のない `id` 名を使用する場合は `id` 名と N_{out} に関連がないため対応関係が把握しづらい．ソースコード中に類似した `id` 名が多用されることによっても可読性は低くなる．

`E:not(F)` セレクタ　特定の要素だけを例外的にセレクタの対象から除くための方法に `E:not(F)` セレクタがある．`E` と `F` は任意のセレクタである．しかし，`F` には結合子を用いることができないためパスセレクタを使用できない．よって，本手法ではこのセレクタを用いずに，打ち消し CSS 規則を追加する．上述の `id` の多用による欠点を許す場合には，`E:not(F)` セレクタを用いることで打ち消し CSS 規則を追加せずにスタイルの意図しない適用を防ぐことができる．

3.2.2 プロパティ値

切り出し対象である SN_{all} の各ノードの算出スタイルの値を切り出しの前後で比較する．値が切り出しの前後で変化した場合は切り出し前の算出スタイルの値をプロパティ値として与える．また，`width` プロパティと `height` プロパティは値の指定がない場合に`"auto"`が値として設定される．`"auto"`はブラウザによって自動で値が計算されるため，切り出しの影響を受ける場合がある．そのため，`width` プロパティと `height` プロパティの指定がなく，切り出しの前後で値が変化した場合にも `width` プロパティと `height` プロパティに絶対値を設定する調整用 CSS 規則を生成する．調整用 CSS のセレクタにはパスセレクタを用いる．

4. 実装

3 章で述べた提案手法を Google Chrome 拡張機能として実装した．本ツールは Google Chrome のバージョン:`64.0.3282.119 (Official Build) (64 ビット)` で動作を確認した．

Google Chrome 拡張機能は Google が提供する Web ブラウザ Google Chrome の機能を追加するためのプログラ

図 9　本ツールのユーザーインターフェース

ムである．拡張機能はブラウザのツールバーにアイコンを表示し，それをクリックすることで拡張機能のポップアップページを表示することができる．拡張機能では閲覧している Web ページに対して JavaScript のソースコードを挿入することができ，これをコンテントスクリプトと呼ぶ．コンテントスクリプトでは DOM API を用いた操作を行える．また，画面に表示されないバックグラウンドページがあり，常に実行されるような処理などを記述できる．これらのページは独立しており，それぞれ異なるライフサイクルを持つ．これらのページ間の通信を行うための機構が用意されている．

提案手法では DOM API を利用するため，切り出しに関する処理をコンテントスクリプトで実装した．本手法のスタイル維持のためのコード変換を行うかどうかなどの設定を変更する UI をポップアップページで実装し，画像などのリソースの取得はバックグラウンドページで行う．通常，ブラウザでは画像などのリソースの取得は同一オリジンポリシーにより制限されており，クロスドメイン通信が行えない．同一オリジンポリシーとは，ドキュメントと異なるオリジンのリソースにはアクセス出来ないようにするためのセキュリティ上の制約である．コンテントスクリプトではクロスドメイン通信が行えないが，拡張機能ではクロスドメイン通信を許可できるためバックグラウンドページでリソースの取得を行っている．

図 9 に実装した本ツールの画面を示す．Web ページ右下に常駐するフローティングメニューを表示し，次の機能を提供する．

■**領域選択**　切り出し可能な HTML の要素にマウスカーソルを乗せることでその要素の領域を囲む枠を表示する．クリックにより領域を選択する．選択する領域は DOM ツリーのノードに対応している．

■**切り出し**　選択した領域の HTML/CSS の切り出しとスタイル維持のためのコード変換を行う．

■**ローカルへ保存**　切り出された HTML/CSS およびリソースを保存する．

5. 実験・評価

5.1 機能テスト

CSS3 にある 42 種類のセレクタに対する機能テスト用 Web ページを作成した．本ツールを用いて各セレクタによってスタイルが適用された HTML の要素に対して切り出し操作を行い，スタイル崩れが生じないか機能テストを行う．切り出しの前後で要素の CSS プロパティ値を比較し，スタイルが維持できた CSS セレクタの種類を評価とする．

5.2 評価と考察

機能テストの結果は，42 種類のうち 40 種類の CSS セレクタにおいてスタイルの維持が確認できた．スタイルの維持ができなかったセレクタは疑似クラスの `E:target` と `E:checked` である．これらのセレクタは外的要因によって対象となる要素が変化する．原因とスタイル維持のための方法について考察する．

`E:target` セレクタ

このセレクタは URI のフラグメント識別子と同じ `id` 属性の値を持つ要素にスタイルが適用される．フラグメント識別子と同じ `id` 属性の値を持てば適用がされるため`:target` と記述することができ，`E:target` と記述することで要素 `E` に限定できる．例として，URI が `http://sample/index.html#foo` であるときフラグメント識別子は `foo` である．以下の HTML ソースコードの a タグのようなページ内部へのリンクをクリックすることで URI フラグメントが URI に付与され，リンク先である `h1` タグまで画面がスクロール移動する．

```
1 <a href="#foo"> go to Header </a>
2
3 <h1 id="foo"> Header </h1>
```

フラグメント識別子 `foo` を持つときに上記の `h1` タグにスタイルを適用するような CSS ソースコードを次に示す．

```
1 h1:target {
2     color:red;
3 }
```

URI フラグメントはユーザーがリンクをたどったときに付与されるため，HTML/CSS では再現ができない．JavaScript を用いて切り出し時のフラグメント識別子を持つ URI に遷移することで実現できると思われる．別の方法として，切り出し時に URI フラグメントを持つときは，1 行目の `:target` の記述を削除し，以下のように CSS ソースコードを変更することでスタイルを適用できると思われる．しかし，リンクをたどったときだけにスタイルを適用するという本来の機能は失われてしまう．

```
1 h1 {
2     color:red;
3 }
```

E:checked セレクタ

このセレクタは input 要素が持つ状態によってスタイルが適用されるかが決まる．input 要素の type 属性の値が"checkbox"であるとき，ユーザーがチェックボックスをクリックすることで状態が変化する．状態の初期値は checked 属性で設定できるが，動的に変化する状態は DOM ノードが保持しており HTML と CSS のソースコードにはその状態が反映されない．そのため HTML/CSS の切り出しではその状態を維持することができない．切り出し時のスタイルを維持するためには，input 要素の切り取り時の状態を再現することで E:checked セレクタのスタイル適用を行う必要がある．切り出し時に input 要素の状態を DOM ノードから取得し，input タグの checked 属性で初期値として与えることで実現可能だと思われる．

実験ではスタイル崩れを確認できなかったが，外的要因に影響するセレクタには:checked，:target，:active，:enabled，:disabled，:hover，:focus，:link，:visited などがある．

切り出し時の外的要因を再現できる場合であれば外的要因に依存するスタイルを維持することができると考えられる．しかし，切り出し時の外的要因の再現は，HTML/CSS のテンプレートとしての再利用を目的としたときに不都合である場合がある．再利用の目的によって維持すべきスタイルかどうかを考える必要がある．

6. おわりに

6.1 まとめ

本研究では，Web ページからユーザーが指定した領域を HTML と CSS の形式のままスタイルを維持して切り出す手法を提案した．HTML に依存したスタイル記述が，切り出し後のスタイル崩れの原因となる．これを防ぐための CSS コード変換方法について述べた．本手法を Google Chrome 拡張機能で実装し，CSS3 で定義されている 42 種類の CSS セレクタに対する機能テストを行い，40 種類のセレクタでスタイルの維持を確認できた．スタイルの維持ができなかったセレクタは外的要因に依存しており，これに対するスタイルの維持の方法について考察した．

7. 関連研究

本研究では，静的 Web ページを対象に Web ページの表示を変えることなく切り出す手法を提案した．張らは，JavaScript による動的な Web ページを対象にユーザーが指定した部分をテンプレートとして抽出する手法を提案している [3]．JavaScript のスライシングには Weiser の手法を用いている [4]．張らの手法では指定部分の HTML とそこで使用されている CSS を抽出する．しかし，ソースコードの抽出により HTML の構造が変化するため CSS のスタイルの適用箇所も変化してしまうが，そのための対応について考慮されていない．本研究ではそのためのスタイル維持のコード変換を行った．

中村が開発したツールは Web ページからユーザーが指定した領域を抽出できる [5]．これにより抽出した Web ページは中村らが提案する P2P を利用したウェブコンテンツ共有システムで活用されている [6]．これはウェブコンテンツの保存を目的としているという点で本研究と目的が異なる．

7.1 今後の課題

今後の課題としては次の 3 点があげられる．

外的要因に依存するスタイルの維持の方法の検討 5.2 節で外的要因によって対象が変化するセレクタとそのスタイル維持のための方法を述べた．再利用に適したスタイル維持についての考察やツールとしての実装方法について検討と実験を行う．

本手法では対応できない場合の調査と改善 実際の Web ページに対しての実験を行い，本手法では維持できないスタイルを調査する．それらのスタイル維持のための方法について検討し，ツールとして追加実装する．

JavaScript のスライシング 提案手法では主にスタイルの維持を目的としたコード抽出と変換を行った．閲覧時のスタイルをそのまま取得することを目指しており静的な Web ページを想定している．そのため JavaScript の抽出は行っていない．今後，Web ページとしての再利用性をより高めるために JavaScript の抽出についても検討する必要がある．

謝辞 本研究は科研費 15K00488 の助成を受けたものである．

参考文献

[1] W3C. Selectors level 3. 2011/9/29.
[2] W3C. W3C DOM4. 2011/11/19.
[3] 張恭瑞, 高田眞吾. Web アプリケーションの UI 開発支援のためのテンプレートの抽出手法. コンピュータ ソフトウェア, Vol. 33, No. 1, pp. 1_150–1_166, 2016.
[4] Mark Weiser. Program slicing. In *Proceedings of the 5th international conference on Software engineering*, pp. 439–449. IEEE Press, 1981.
[5] 中村聡史. Webox. http://webox.sakura.ne.jp/software/webox/.
[6] 中村聡史, 塚本昌彦, 西尾章治郎. コンテンツ流通制御を考慮したウェブコンテンツ共有システムの実現. 情報処理学会論文誌, Vol. 45, No. 1, pp. 74–83, 2004.